Open access Journal International Journal of Emerging Trends in Science and Technology
Impact Factor: 2.838 DOI: http://dx.doi.org/10.18535/ijetst/v3i03.13

Native Client: A Sandbox Technology

Authors
Geetika Kapil, Lata Bagle, Gohila Khatun, Mrs Ruchi Tawani

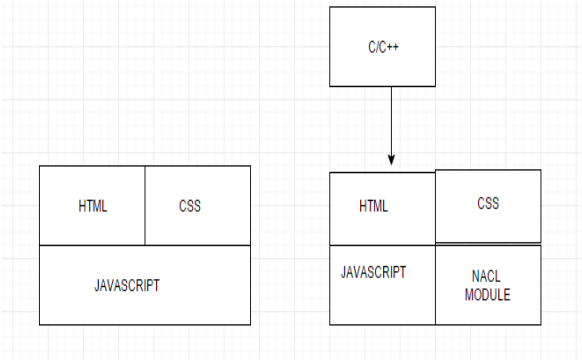
Abstract
Native Client is a sandbox technology, which run C and C++ compiled code in the browser in an efficient and secure manner. To make this technology go independent of any architecture, portable native client let developers to compile their code with Ahead Of time (AOT) translation. It brings the native code to the web without the loss of the security and portability of the web.

INTRODUCTION

Native Client is an open source technology that expands web programming beyond Java Script, enabling us to enhance application without the loss of security .The developers can work in their preferred language. This paper digs in the advantages of this technology, common use cases, how to use it for porting Perl modules etc. The Google has implemented the open source Native Client project in the chrome Browser for Mac, Windows and Linux.

The structure of the web application

An web application consists of the languages like HTML,CSS ,JavaScript and a NaCl module written in a language supported by the SDK. The SDK is there for C and C++ , we have mainly worked on the Perl language SDK.



A web application with and without Native Client

Why use Native Client?

It helps to run the compiled code in the browser at near native speeds. It helps to harness the client's computational resources for 3D, CAD modeling, client-side data analytics.

It gives the compiled code the same level of portability and safety as the JavaScript.

Benefits of Native Client

Below are the advantages of the native client

1. Graphics, audio:-We can easily play 2D and 3D games. Audio, respond to the mouse. Even the application can run on multiple threads and access memory directly without installing any plugins.

2. Portability:- It is possible now to write your application once and run them on multiple operating systems architecture.

3. Easy migration path to the web:- Its support to the C and C++ modules make it easy for native client to transit the desktop applications to the web.

4. Security: Double sandbox technology prevents the system from malicious or buggy applications. The model offers the safety to the traditional web applications without the sacrifice of the performance .The user need not even install the plugins required.

4. Easy migration path to the web:-It support of C and C++ make it easy for native client to transit the desktop applications to the web.

5. Security: Double sandbox technology prevents the system from malicious or buggy applications.

Without allowing the user to install the plugin and without the sacrifice of the performance the model offers the safety of traditional web application.

6. **Performance:** It allows to harness the all available CPU cores via a threading API and that makes it to run at 5% to 15% of a native client. This allows to run the games in the browser.

Common use cases

Following are the use cases defined:-

Existing software components: We need not write the code that already works which helps us to repurpose existing C and C++ software in web application. Moreover, HTML 5 can also be explored.

Legacy desktop applications: We can easily port and recompile the exciting code on the computation engine of our application.

Heavy computation in enterprise applications: To avoid the unencrypted data to escape out of the network. Native client helps us to run complex algorithmic algorithms directly in the browser.

Multimedia applications: In the Native Client the codes which process the sounds, images and movies can easily be added in the form of the module.

Games: Web applications can run at close to native speeds, they can reuse existing multithreaded/multicore C/C++ code bases. They easily take in control of the low –latency audio, OpenGL ES and networking APIS with programmable shaders.

The AI modules make it possible for sophisticated web games to run. It is easy now to run the unchanged application on various systems.

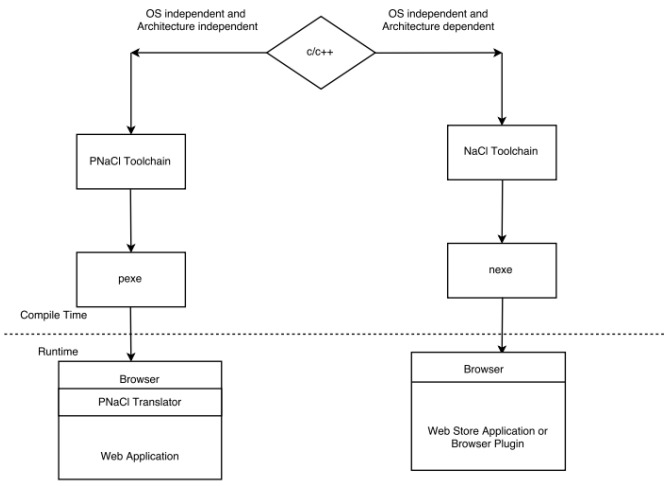
How Native Client works:

Native Client is consists of:

Tool chains: These are a set of a compiler, linkers that transform C/C++ code into portable Native Client modules .

Runtime components: These are the components embedded in the browser that allow the execution of Native Client modules more securely and efficiently.

The diagram given below shows how the components react



The Native Client tool chains and their outputs

Tool chains

It is consist of a compiler, a linker, an assembler and other tools which is used to convert C/C++ source code into a module which is easily loadable by a browser

Following are the two tool chains which NaCl provides:-

- In the left of the diagram above is Portable **Native Client** (PNaCl, pronounced "pinnacle"). A single portable (.pexe) module is produced by LLVM based tool chain.
- At runtime an ahead-of-time (AOT) translator, built into the browser, translates the .pexe into native code for the client machine.

The right side of the same diagram shows **(non-portable) Native Client**. Multiple architecture-dependent (.nexe) modules that are packaged into an application are produced by a GCC based tool chain. Based on the architecture of client machine at runtime the browser decides which .nexe modules to be loaded.Most of the applications can run with a PNacl tool chain.

Security

To provide the security for the client implementing native client code we ensure the following things:-

-The NaCl sandbox sees to it that code access system resources only through safe, whitelisted APIs, and operates them without attempting to interfere with other code running either within the browser or outside it

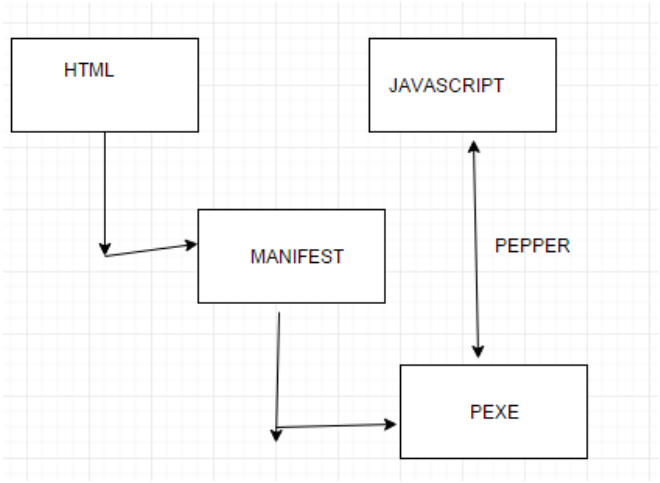
-The validator checks the code and data patterns before running to make sure they are safe.
-With restricted permission the native client module always executes in a process with restricted permission. The process can interact with the outside world through defined browser interfaces. Because the native client is a combination of Nacl Sandbox and the Chrome sandbox it is sometimes said as double sandbox technology.

Portability
To compile C/C++ source code to portable bit code executable (.pexe) PNacl employs compiler technology.

PNaCl bit code is an OS- and architecture –independent format that can be embedded in web application and distributed on the web. PNaCl translator runs .pexe modules. The translator compiles a .pexe to a .nexe and executes .nexe in Native Client sandbox. The translator uses caching to avoid the recompiling the .pexe if it was already compiled on the client browser. The .nexe modules are not allowed to be distributed on the open web because it is architecture –specific but can easily be compiled in the browser.

Structure of a web application
Following are the set of files in the Native Client:-
HTML and CSS: Through the embed tag the HTML file tells the browser where it can find the .nmf file as shown below:-
<embedname="mygame"src="mygame.nmf"type="
application/x-pnacl"/>
Manifest: The manifest finds the respective module to load and also specifies options.
. For example, "mygame.nmf" might look like this:
{...
...
"url":"mygame.pexe",
}

.pexe (portable NaCl file):It is the compile native Client module using the Pepper API,it is also the medium between the JavaScript and other browser resources.



Structure of a web application
Pepper plug-in API
Pepper is an open source, cross platform C/C++ API for web browser plug-in to access system-level functions in a safe and portable way .It allows a C/C++ module to communicate with the hosting browser.
It provides analogous API's of OS-level calls that module can use. It can be used to gain access to the full browser capability, including

- We can talk to the JavaScript code in our application from the C++ code in our Nacl module.
- Doing file I/O.
- Playing audio.
- Rendering 3D graphics.

It includes both a **C API** *and a* **C++ API**. *The C++ API is a set of bindings written on top of the C API.*

- Native Client comes in two flavors.

Portable Native Client(PNaCl):-Commonly known as pinnacle it runs single, portable (.pexe) executables and is generally available in implementation of the Chrome. For the purpose of client hardware, a translator built into the chrome translates the .pexe into native code .It can be hosted from any web server easily.
Native Client (NaCl): It runs architecture-dependent (.nexe) modules, which are packed into an application. based on the architecture of the client machine. The browser decides which .nexe to load.
Native Client (NaCl)
By the use of advanced Software Isolation technique the Native Client enables the execution of native code securely and efficiently inside the web

application .It also allows us to harness a client machine computational power to the fullest extent by running compiled C and C++ code at near – native speed. It also exposes CPU full capability, including SIMD vectors and multiple-core processing with shared memory.

Usually HTML, CSS, JavaScript bundles were hosted on a server and run in a web browser which is generally working on all platforms for years. Executables which are architecture- specific are not a good fit for distribution on the web.

Portable Native Client (PNaCl):-It solves the problem of portability by splitting the process of compilation into two parts:-Compilation of the source code to the bit code executable (.pexe)

-translation of the bit code to a host-specific executable as the module loads in the browser but prior to any code execution

This helps Native Client to align with the existing open web technologies such as JavaScript. The user machine is able to run a .pexe module as a part of an application (along with HTML, CSS, and JavaScript)

With the help of PNacl we can generate single .pexe rather than multiple .nexe. The .pexe uses an abstract, architecture and OS-independent format. This is the sole reason why it does not suffer from the problem of portability.

Nacl and PNacl have the same level of security,but pinnacle can be more efficient on some operating systems than the others. If an existing architecture is enhanced, the .pexe doesn't need to be recompiled and future versions of hosting environments will not have any problem executing the .pexe, even on new architectures.

When to use P NaCl: The only way to deploy Native Client modules without the Google Web Store is the pinnacle. Without installing browser plug-in, Chrome can easily translates. pexe modules and support their use in web applications. By using a P Nacl tool chain, we can take the advantage of the conveniences of PNacl, such as not having to explicitly the application for all supported architectures.

When to use NaCl

We can use NaCl if any of the following applies to our application:

- Our application requires architecture-specific instructions such as, for example, inline assembly.
- Portable SIMD Vectors tries to offer –high performance equivalents which are portable.

Our application uses dynamic linking. It supports static linking with the P Nacl port of the newly C standard library.

Something are not supported in P Nacl like dynamic linking and glibc. The work of supporting dynamic linking is already in place.

Taking the address of a label for computed go to or nested functions are not supported by GNU extensions.

Porting Perl

PNaCl already has support for C and C++, and virtual machine such as JavaScript, Lau, Python and Ruby.

We are working on LLVM bit code so that this language can target on .This will also help to make sure that the language virtual machine APIs work well on the Net platform.

For porting Perl we require following prerequisites.

Prerequisites

The minimum requirements for using web ports are:

- python 2.7
- python-dev
- gclient (from depot tools)
- Native Client SDK

The following tools are required to build the packages from the source .These tools are required by the build scripts .

- make
- curl
- sed
- git

To work on ports we would require this all

- pkg-config
- autoconf, automake, libtool
- cmake
- texinfo
- gettext

- libglib2.0-dev >= 2.26.0 (if you want to build glib)
- xsltproc

We can use homebrew to install these by executing the following commands:-

$ brew install autoconf automake cmake get text libtool pkg-config

On 64-bit Ubuntu/Trusty systems we need to install

- libssl-dev:i386
- zlib1g-dev:i386

These required for the builds system for native Python modules which relies on 32 bit host platformbuilds of Python. This in ahead linked to the development versions of zlib and libssl

On older Debian/Ubuntu systems these packages were known as:

- libssl1.0.0:i386
- lib32z1-dev

The tools listed below are also needed in order to run some of the binaries in 32-bit system. In the Nacl SDK.

- libglib2.0-0:i386
- libstdc++6:i386

How to Checkout

Follow the following steps to correctly check the web ports

1)Create a directory:

$ mkdirwebports

$ cdwebports

2)Create a .gclient Configuration:

$ gclientconfig --unmanaged --name=src \

https://chromium.googlesource.com/webports.git

https://chromium.googlesource.com/webports/

https://github.com/adlr/naclports

3)Sync the code and dependencies:

$ gclient sync --with_branch_heads

The pepper canary is used by the master branch .We need to switch the corresponding pepper X branch for older SDK.

e.g:

$ cdsrc

$ git checkout -b pepper_49 origin/pepper_49

$ gclient sync

Building

The path to set the NACL_SDK_ROOT environment variable on the top directory of the Native Client SDK is absolute.

To build one or more packages the top level make file is used as a quick way to build on one or more packages..Taking an example, make libvorbis will help in building libvorbis and libogg .make all will help in building all the packages. There are 4 possible architectures that NaCl modules can be compiled for: i686, x86_64, arm, pinnacle. The web ports build system can be built just a single time. By setting the NACL_ARCH environment variable we can have our choice as seen below:-

$ cdsrc

$ NACL_ARCH=arm make openssl

There is more than one tool chain is available for some of the q=architectures. We can have a choice between new lib and g-libc for x 86 platforms. By specifying the Tool chain environment variable we can default tool chain to pinnacle.

$ NACL_ARCH=i686 TOOLCHAIN=glibc make openssl

By executing the command makeall.shscript we can build a certain package for all architectures and all tool chains.

e.g.:

$ ./make_all.sh openssl

In order to make use of the libraries, headers and libraries are installed into the tool chains directly so there is no need to add extra –l or –L.

We can follow the following path to see the source code and build output for each package.

out/build/<PACKAGE_NAME>

To make the debug packages set NACL_DEBUG=1 or PASS—debug to the web ports script. They are by default available in release configuration.

Note: Each package has its own license. Please read and understand these licenses before using these packages in the projects.

For Windows : The all scripts are to be launched from a Cygwin shell as they are written in bash.Webports are only rested in Linux and windows framework so as YMMV

Binary Packages

By default web ports will attempt to install binary packages rather than building them from source. The build bots produce the binary packages and store them in Google cloud storage. The index of current binary packages is stored inlib/prebuilt.txt and this is currently manually updated by running build tools/scan_packages.py. The package will always be built from source if the package version does not match the version. To build a package from source we can pass-from –source or FROM_SOURCE=1 on the make command line.

Emscripten Support

The build system contains very early alpha support for building packages with Emscripten. To do that we require the Emscripten SDK to be installed and configured (with the Emscripten tools in the PATH). To build for Emscripten builds with TOOLCHAIN = emscripten.

Running the Examples

Out. Publish contains all the Applications/Examples that build runnable web packages .We will need a web server to run them in chrome by executing the following command :

$make run

After a local web server to start serving the content we can view the result at https://localhost:5103 to have a look at the output.

Conclusion

Native Client uses software which provides fault isolation and a secure runtime to direct system interaction and side effects through interfaces handled by the Native Client. Native Client gives operating system portability of binary code while supporting performance-oriented features which are generally absent from web application programming environments, instruction set extensions such as SSE,Thread support and use of compiler intrinsic and hand-coded assembler. We combine these properties in an open architecture that encourages community review and 3rd-party tools.

As an application platform, the modern web browser provides together a remarkable combination of resources, high productivity, programming languages such as JavaScript, including seamless access to Internet resources and the richness of the Document Object Model (DOM) for user interaction and graphic presentation. While these strengths put the browser in the forefront as a goal for new application development, it remains handicapped in a critical dimension: computational performance. Thanks to Moore's Law and the zeal with which it is observed by the hardware community, despite this handicap many interesting applications get adequate performance result in a browser. But there remains a set of computations that are generally not suitable for browser-based applications due to performance constraints, for example: simulation of computational fluid-dynamics, Newtonian physics and high-resolution scene rendering.

References

1. Google's Native Client engineering team has published the following papers about Native Client technology:
2. http://cacm.acm.org/magazines/2010/1/55768-native-client-a-sandbox-for-portable-untrusted-x86-native-code/fulltext
3. http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.ieee-000005207638
4. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5207638